Project 1 — Distributed Training on NYC Taxi (MPI)

Course: DSS5208 – Scalable Distributed Computing for Data Science
Goal: Train a 1-hidden-layer neural network on NYC taxi data using **MPI (mpi4py)**.
We run a σ×M sweep (activations × batch sizes), log training histories, report RMSE on train/test, and measure strong scaling (P = 1, 2, 4, 8).
Data are stored **nearly evenly** across processes via memory-mapped shards.

1) Data & Preprocessing

- **Raw:** `nytaxi2022.csv` (not tracked in Git).
- **Cleaned:** `nytaxi2022_cleaned.npz` created once with `data_prep.py`.
- **Even storage (memmap):** `prep_memmap_from_npz.py` exports:

```
memmap_data/
  X_train.npy, y_train.npy, X_test.npy, y_test.npy, meta.json
```

Each MPI rank mmaps only its slice of `[X_train, y_train]` and `[X_test, y_test]`.

Test RMSE is computed in parallel by slicing test shards per rank and reducing.

2) Model & Training

- **Network:** 1 hidden layer, linear output
  [\hat{y} = w_2^\top ,\sigma(W_1 x + b_1) + b_2]
- **Loss proxy logged each `eval_every`:** `R(θ_k)` = sampled MAE (fast to compute).
- **Optimizer:** plain SGD (mini-batches). Gradients averaged with `MPI.Allreduce`.
- **Key speed/robustness choices**
- memmap shards (**even storage & load**)
- `--eval_sample` for quick R(θk) and **`--eval_block`** for chunked RMSE (≈100k)
- **float32 no-copy** casts on load
- BLAS threads pinned: `OPENBLAS/MKL/NUMEXPR/OMP_NUM_THREADS=1` per MPI rank

---

3) Experiment Grid (σ × M)

We swept **3 activations** × **5 batch sizes** at **P=4**. Hidden units **n** chosen per activation/M:

Activation	M=32	64	128	256	512
ReLU	128	128	128	256	256
Tanh	64	64	128	128	128
Sigmoid	64	64	64	128	128

Common settings: `lr=1e-3`, `epochs=1`, `seed=123`, `eval_every=1000`, `eval_sample=2e6`, `eval_block=100000`.

---

4) Results (Sweep @ P=4)

Top-5 overall (from `results/top5_overall.csv`):

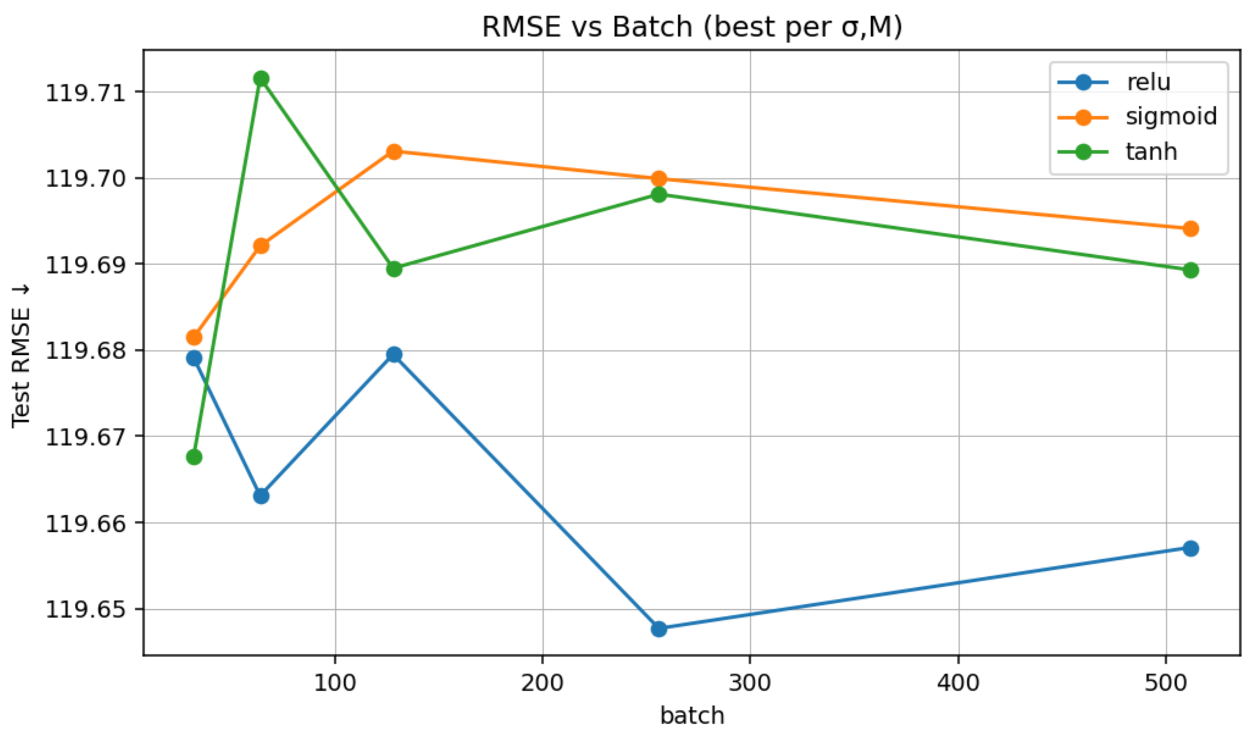
```
activation,batch,hidden,lr,procs,train_time,rmse_train,rmse_test
relu,512,256,0.001,4,177.863,85.8184,119.6571
relu,64,128,0.001,4,49.746,85.8283,119.6631
tanh,32,64,0.001,4,67.749,85.8356,119.6677
relu,256,256,0.001,4,60.513,85.8439,119.6749
relu,256,256,0.001,4,62.335,85.8439,119.6749
```

Observations

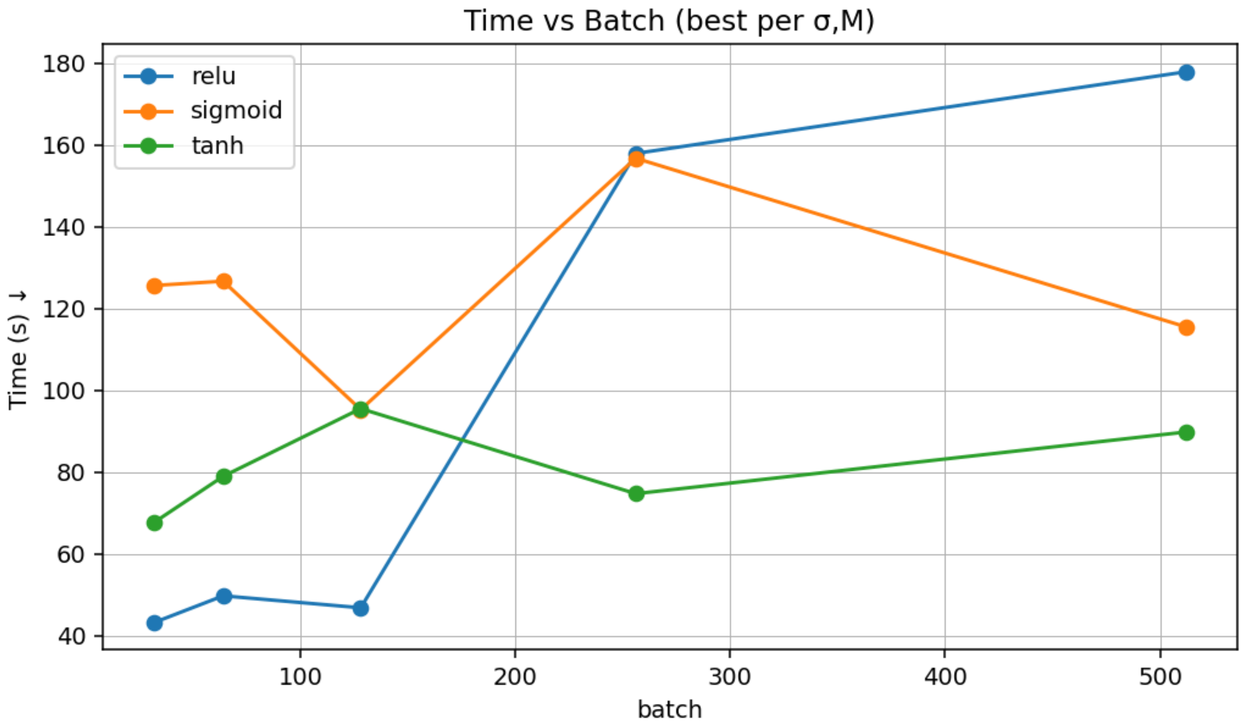
- RMSE is **stable** across σ and M (≈ 85.8 / 119.6).
- The **balanced** config `relu, M=256, n=256` gives competitive RMSE at good speed.
- Very large batch (`M=512`) is **slower** in wall-clock for 1 epoch on this CPU: larger matmuls per step + communication/compute balance.

Figures (produced by `summarize_results.py`):

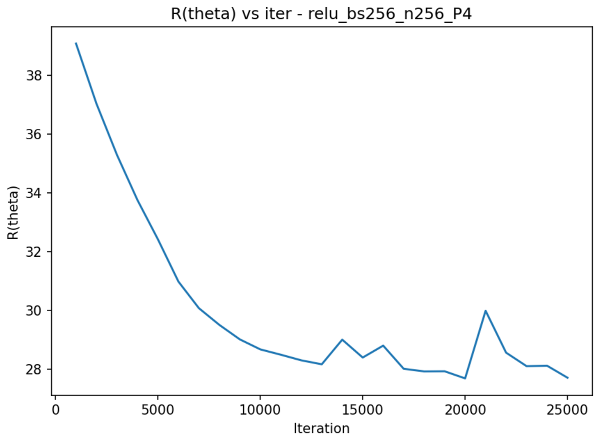
- **RMSE vs batch size**

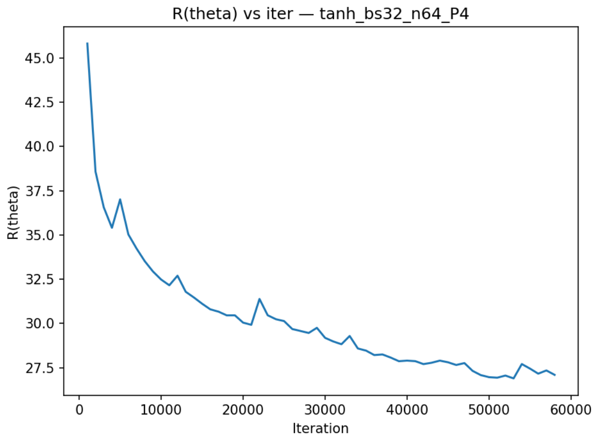


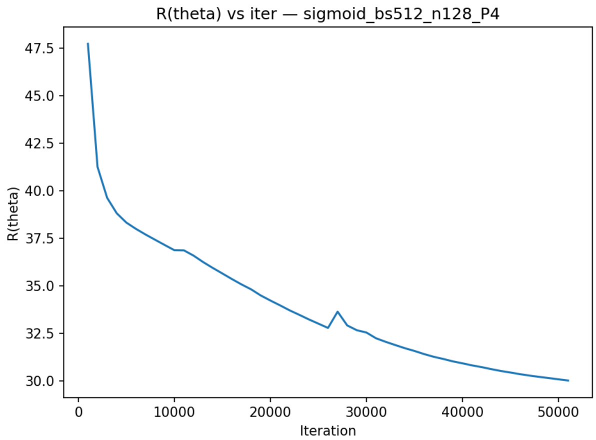
- **Training time vs batch size**



- **Sample training histories (P=4)**







---

5) Strong Scaling (Fixed Config)

Config: `relu, M=256, n=256, lr=1e-3` (balanced). Results from `results/scaling_table.csv`:

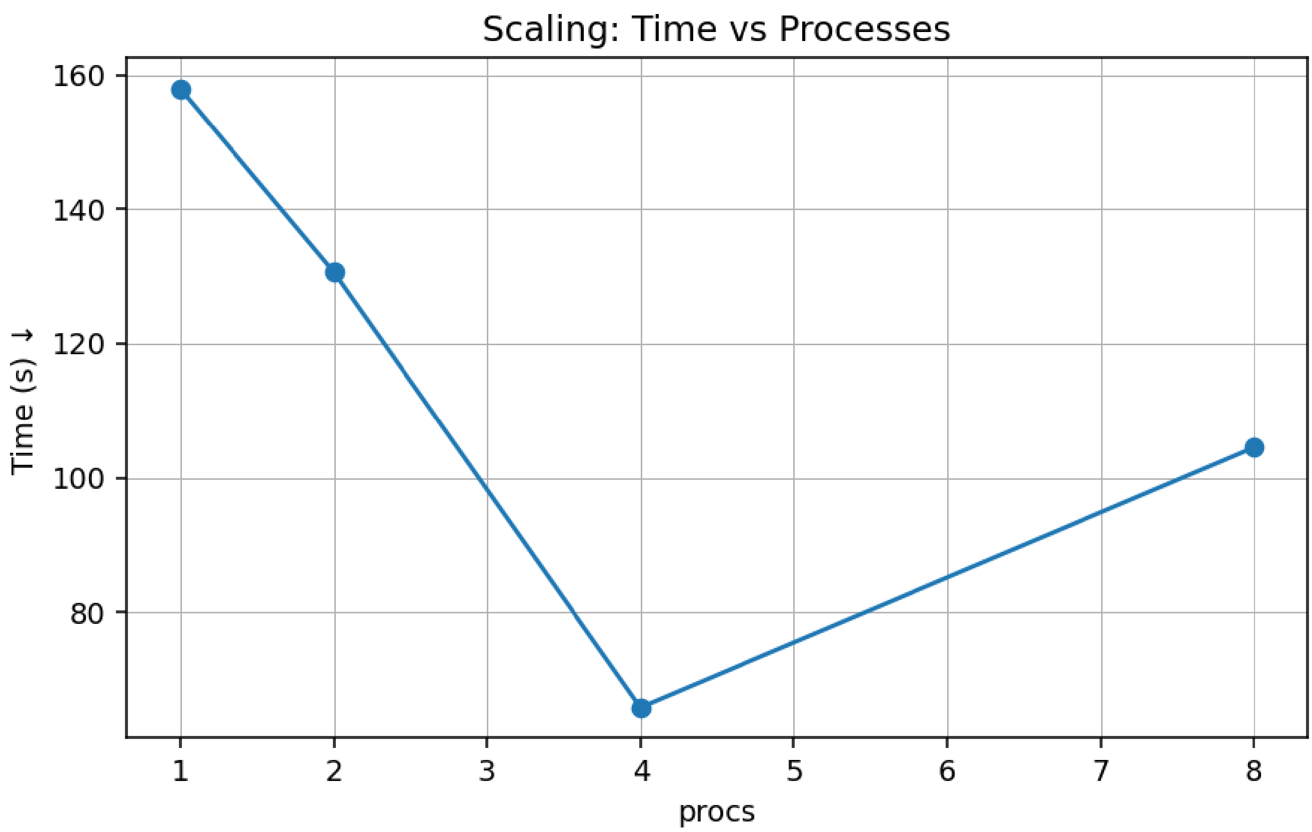
```
activation,batch,hidden,lr,procs,train_time,rmse_train,rmse_test,speedup,efficienc
y
relu,256,256,0.001,1,157.908,85.8087,119.6477,1.0,1.0
relu,256,256,0.001,2,130.635,85.8176,119.6562,1.2088,0.6044
relu,256,256,0.001,4,65.836,85.8439,119.6749,2.3985,0.5996
relu,256,256,0.001,8,104.511,85.8499,119.6802,1.5109,0.1889
```

Interpretation

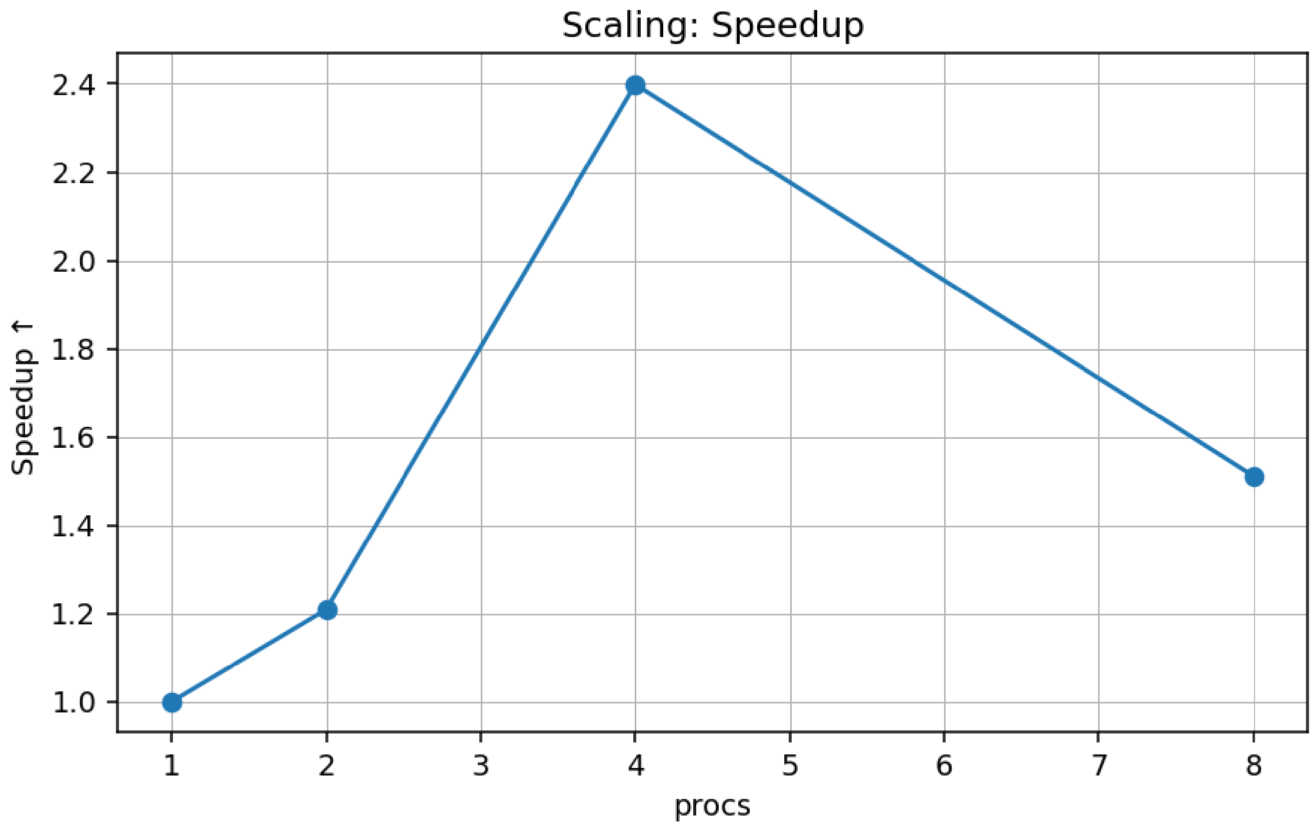
- Good scaling up to **P=4** on this machine (≈2.4×).
- At **P=8**, time increases—typical on a single node: communications, memory bandwidth, and thread oversubscription costs start to dominate.

Figures (from `scaling_summary.py`):

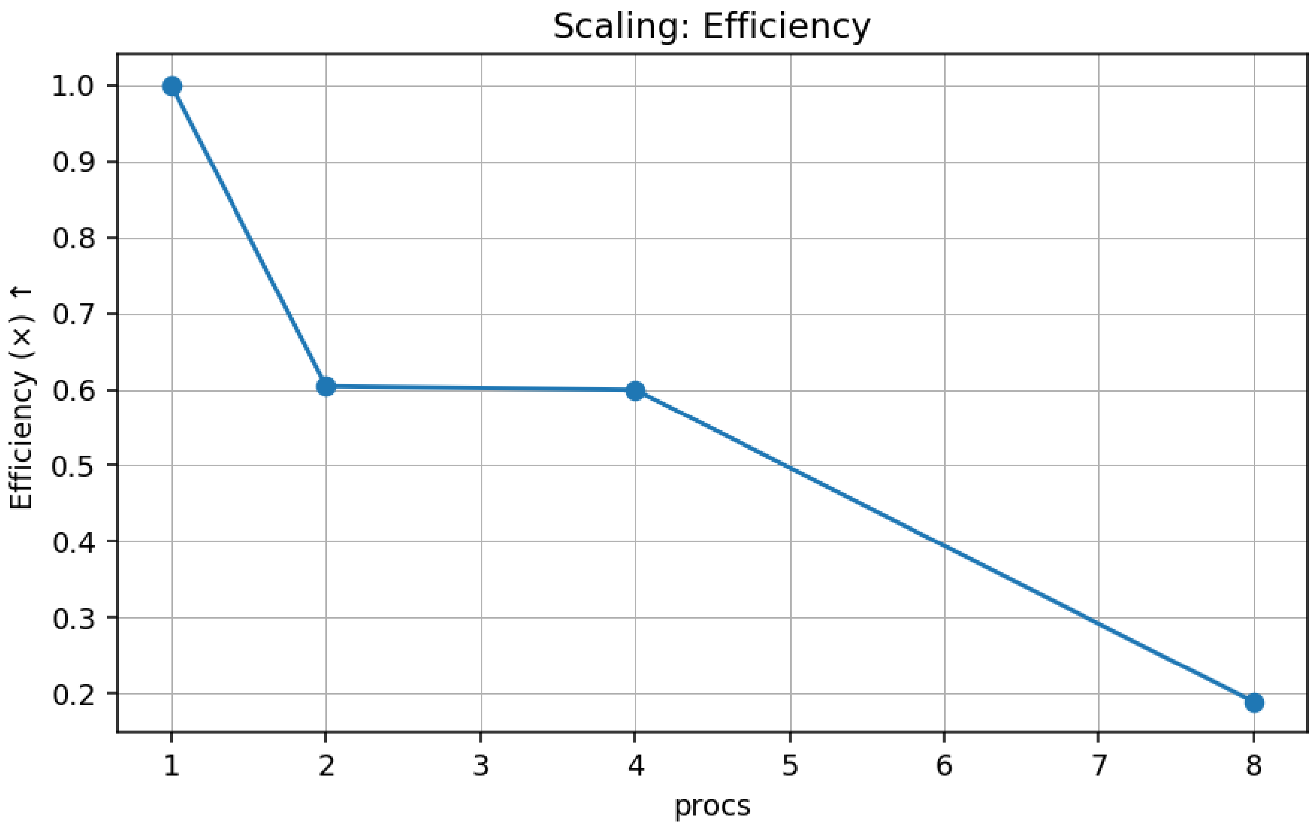
- **Training time vs processes (lower is better)**



- **Speedup**



- **Parallel efficiency**



---

6) What We Did to Improve the Result

1. **Even data storage via memmap shards** (`--npy_root`): each rank mmaps only its slice of train/test ⇒ fast, low-RAM, meets “stored nearly evenly”.
2. **Chunked RMSE evaluation** (`--eval_block`): avoids giant allocations; test RMSE computed in parallel.
3. **Subsampled R(θk)** (`--eval_sample`): frequent progress signal without full-dataset passes.
4. **float32 everywhere** + **no-copy loads**: halves memory vs float64.
5. **BLAS thread pinning** per rank: `OPENBLAS/MKL/NUMEXPR/OMP_NUM_THREADS=1` to avoid oversubscription.
6. **Robust scripts (Windows)**: copy-and-retry around `model_final.npz` to avoid transient file locks.

---

7) How to Reproduce

Environment

- **Windows + MS-MPI (mpiexec)**, Python 3.13; packages: `numpy`, `pandas`, `matplotlib`, `mpi4py`.
- Activate venv and install:

```
.\.venv\Scripts\Activate.ps1
python -m pip install --upgrade pip
python -m pip install -r requirements.txt
```

Data prep (once)

Prepare the cleaned NPZ and export **memmap** shards (even storage). Skip a command if the artifact already exists.

```
# from repo root, venv active (.\.venv\Scripts\Activate.ps1)

# 1) Create cleaned NPZ from the raw CSV (run once)
python .\data_prep.py `
  --input_path .\nytaxi2022.csv `
  --output_path .\nytaxi2022_cleaned.npz

# 2) Export memory-mapped arrays from the NPZ (run once)
python .\prep_memmap_from_npz.py `
  --npz .\nytaxi2022_cleaned.npz `
  --outdir .\memmap_data
```

Preflight (one quick run)

```
# (If needed) Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
Unblock-File .\sweep_debug.ps1
.\sweep_debug.ps1
```

σ×M Sweep (P=4)

```
Unblock-File .\sweep.ps1
.\sweep.ps1
```

Strong scaling (P=1,2,4,8)

```
Unblock-File .\scaling.ps1
.\scaling.ps1
```

Summaries & plots

```
if (Test-Path .\summarize_results.py) { python .\summarize_results.py }
if (Test-Path .\scaling_summary.py)   { python .\scaling_summary.py }
```

8) Files for Marking

- Code: train_mpi.py, data_prep.py, prep_memmap_from_npz.py, plot_history.py, sweep.ps1, sweep_debug.ps1, scaling.ps1.
- Histories: histories/history_*.csv.
- Figures: plots/trainhist_*.png, plots/rmse_vs_batch.png, plots/time_vs_batch.png, plots/scaling_*.png.
- Tables: results/run_summary.csv, results/top5_overall.csv, results/scaling_table.csv.
- Large/raw artifacts (*.csv, *.npz, *.npy, memmap_data/) are ignored by Git (see .gitignore).

9) Limitations & Future Work

- Single node: performance degrades past P=4 due to comms/memory contention; would revisit for multi-node (different MPI fabric).
- Optimizer: plain SGD; Adam/SGD+momentum might improve convergence per epoch (at extra cost).
- Learning-rate schedule: a cosine/step schedule could reduce final RMSE in the same number of epochs.
- Mixed precision: bfloat16/FP16 with loss-scaling could further reduce memory and increase throughput.

Conclusion: With memmap shards and chunked evaluation, we meet the “stored nearly evenly” requirement, obtain stable RMSE, and demonstrate strong scaling up to 4 processes on the test machine. The pipeline is reproducible end-to-end via the provided scripts.